TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Softwarové rozhraní pro laserový triangulační senzor TLE1

Diplomová práce

M12000213

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Tomáš Němeček**
Vedoucí práce: Ing. Miroslav Holada, Ph.D.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Software interface for laser triangulation sensor TLE1

Diploma thesis

M12000213

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Tomáš Němeček**
Supervisor: Ing. Miroslav Holada, Ph.D.

Liberec 2014

Tento list nahraďte
originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Rád bych poděkoval svému vedoucímu panu Ing. Miroslavu Holadovi, Ph.D. za vedení mé práce a firmě Metralight za zapůjčení senzoru, trpělivost a poskytnutí mnoha cenných rad.

Abstrakt

Cílem této práce je navržení softwarového rozhraní pro laserový triangulační senzor TLE1 firmy Metralight a demonstrace jeho schopností pomocí ukázkové aplikace, která rozhraní implementuje. K řešení byl použit programovací jazyk C# .NET a framework Windows Presentation Foundation. Prvním bodem realizace bylo seznámení se se senzorem a jeho komunikačním protokolem. Druhým bodem byl návrh softwarového rozhraní a jeho realizace, s čímž také souvisí nutnost vytvoření přehledné dokumentace v anglickém jazyce. Posledním bodem řešení bylo vytvoření ukázkové aplikace, která bude na jedné straně demonstrovat schopnosti senzoru a jeho případné využití v rámci Technické univerzity v Liberci a na straně druhé otestuje možnosti navrženého rozhraní. To bylo zároveň otestováno v rámci firmy Metralight, která jej nabídla svým zákazníkům, kteří již nejsou nucení při programování aplikací studovat princip práce komunikačního protokolu senzoru. Výsledky této diplomové práce umožňují významné zrychlení práce se senzorem TLE1 a jeho snadnější implementaci v rámci vlastních systému.

Klíčová slova

laserový triangulační senzor, softwarové rozhraní, programovací jazyk C#, vizualizace dat, komunikační protokol

Abstract

The aim of this work is to design a software interface for laser triangulation sensor TLE1 and demonstration of its capabilities with a sample application that implements the interface. The solution is programmed in C# .NET programming language with Windows Presentation Foundation framework. First point was to study the sensor and its communication protocol. Ssecond point was proposal of a software interface and its implementation which is also linked with creating documentation in English. The final point of the solution was to create a sample application that will demonstrate capabilities of TLE1 sensor and its possible use in the Technical University of Liberec. Proposed interface has been tested by Metralight Company and their customers. They are no longer forced to study principle of sensors communication protocol. Results of this work allows significant simplification of work with TLE1 sensor and easier implementation within own systems.

Key words

laser triangulation sensor, interface, C# programming language, data visualization, communication protocol

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Seznam zkratek

BMP	Bitmap, Formát pro ukládání rastrové grafiky
EEPROM	Electrically Erasable Programmable Read-Only Memory, Paměť umožňující mazání elektronickým pulzem
MSB	Most Significant Bit, Nejvyšší bit
POE	Power Over Ethernet, Způsob přenosu napětí přes ethernetové rozhraní
ROI	Region of Interest, oblast v rámci které senzor zpracovává data
USB	Universal Serial Bus, univerzální sériové komunikační rozhraní
WPF	Windows Presentation Foundation, framework jazyka C# .NET umožňující tvorbu grafických komponent

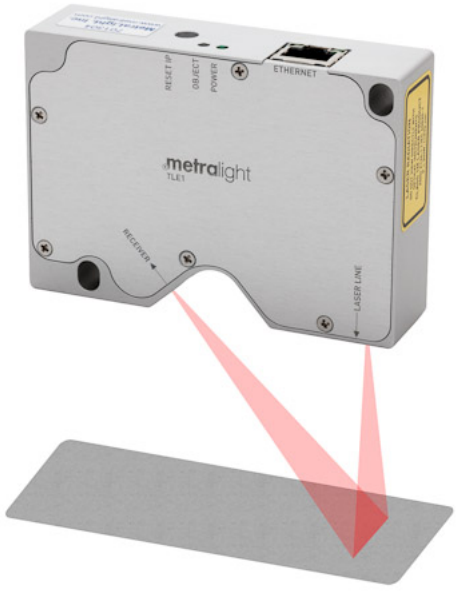
Úvod

Diplomová práce se zabývá problematikou návrhu softwarového rozhraní pro laserový senzor TLE1. Cílem práce je vytvořit prostředek pro jednodušší práci se senzorem bez pokročilých znalostí principu jeho měření či komunikačního protokolu a demonstrovat jeho použití vytvořením ukázkové aplikace, která jej implementuje a zároveň naznačí možná využití senzoru v praxi. Senzor TLE1 byl vybrán na základě předchozí spolupráce se společností Metralight, která senzor vyrábí a která potřebovala svým zákazníkům nabídnout vhodné řešení pro urychlení zařazení senzoru do jejich systémů či výrobních linek.

Následující kapitola shrnuje vlastnosti a princip měření senzoru TLE1 a stručně představuje jeho komunikační protokol. Jsou zde také nastíněna možná úskalí při vytváření softwarového rozhraní. Třetí kapitola se zabývá vlastním vytvářením rozhraní. Je zde popsán návrh rozhraní, řešení různých způsobů připojení k senzoru, zpracování několika typů měřených dat a práce s daty v paměti. Závěrem kapitoly je podrobně popsána třída, která celý senzor reprezentuje. Čtvrtá kapitola představuje použití vytvořeného rozhraní v rámci ukázkové aplikace, která současně naznačuje i možné způsoby vizualizace měřených dat a možnosti využití senzoru jak v rámci Technické univerzity v Liberci, tak v praxi.

2 Senzor TLE1

TLE1 je laserový triangulační senzor vyrobený společností Metralight sloužící k velmi přesnému měření vzdálenosti od objektu a jeho výšky. Na rozdíl od jiných podobných senzorů nepracuje pouze s jedním bodem vyslaným k objektu, ale na objekt vyšle laserovou čáru, kterou následně zachytí pomocí CMOS čidla s rozlišením 1280x1024 px. Senzor je tak schopen přesněji změřit vzdálenost od objektu (vzhledem k většímu množství bodů), určit šířku otvoru v objektu nebo zachytit profil části výrobku.



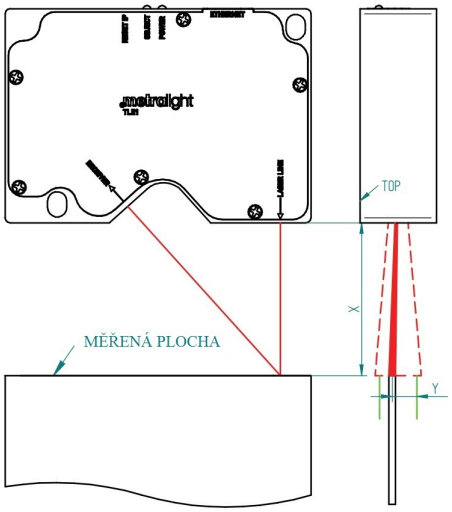
Obrázek 2.1: Senzor TLE1

2.1 Parametry senzoru

Jak již bylo řečeno výše, jedná se o senzor pro měření vzdálenosti a výšky pracující na principu laserové triangulace. Rozlišení tohoto měření dosahuje 1 μm pro měření vzdálenosti a méně než 20 μm pro měření výšky. Oblast měření začíná 35 mm od

hrany senzoru a končí na 65 mm. S okolím senzor komunikuje pomocí ethernetového rozhraní a je možné využít jak TCP, tak UDP protokol. Přes ethernetové rozhraní je senzor současně i napájen použitím POE. Využití POE ale zároveň znamená omezení maximální komunikační rychlosti na 100 Mbit/s. Rozměry senzoru jsou uvedeny v příloze A.

Senzor pracuje ve čtyřech základních režimech měření. V režimu MODE_0 měří pouze vzdálenost (osa X) od objektu průměrováním hodnot uživatelem definovaného množství bodů. V režimu MODE_1 měří kromě vzdálenosti i polohu objektu v ose Y. V režimech MODE_2 a MODE_3 senzor měří vzdálenost od objektu v ose X a v Y ose určuje pozici objektu od pravé resp. levé strany. Manuál Metralight (2012) označuje druhý měřený rozměr jako „height"– tedy výška. Tento pojem je však zavádějící, protože se nejedná o výšku prostorovou, ale výšku ve 2D. Konkrétní ukázku měřených rozměrů při aktivovaném režimu MODE_1 ukazuje obrázek 2.2. V textu dále bude ale pro zachování jednotnosti pojmů tento rozměr označován jako výška.



Obrázek 2.2: Rozměry měřené senzorem

2.2 Uživatelské parametry

Uživatel má možnost ovlivnit práci senzoru nastavením několika parametrů. Parametry se dělí na nevolatilní, které jsou uloženy ve vnitřní EEPROM paměti, a volatilní, které se při každém restartování senzoru nastaví na hodnoty, které jsou uloženy v EEPROM paměti. Změna volatilních parametrů bude mít tedy vliv na práci senzoru jen do doby, dokud je senzor zapnut.

2.2.1 Nevolatilní parametry

Nevolatilní uživatelské parametry jsou v senzoru uloženy na devíti stránkách paměti EEPROM. Na stránce s adresou 255 jsou uloženy následující parametry nastavující ethernetové připojení:

- **IP adresa senzoru**
- **Maska podsítě**
- **Výchozí brána**

Na stránkách 247 až 254 jsou uloženy ostatní parametry, přičemž na každé stránce jsou uloženy ty samé parametry ve stejném pořadí, což umožňuje vytvořit 8 přednastavení, která lze během práce se senzorem přepínat. Parametry jsou uloženy v následujícím pořadí.

- **Peak Size From** – Minimální hodnota signálu, která bude zpracována
- **Peak Size To** – Maximální hodnota signálu, která bude zpracována
- **Line Processing Mode** – Režim zpracování měřených hodnot
- **Default Mode** – Režim měření, který senzor po startu nastaví
- **Offset X** – Přičtení / odečtení hodnoty k měřenému číslu
- **Offset Y** – Přičtení / odečtení hodnoty k měřenému číslu

- **Laser Power** – Výkon laserového vysílače
- **Time Of Integration** – Doba zpracování paprsku na CMOS snímači
- **Threshold** – Hodnota prahu, pod kterým se již data nezpracovávají
- **Average Window** – Nastavení okénkovací funkce
- **Region of Interest** – Nastavení oblasti v rámci které se zpracovávají data
- **Window** – Nastavení oblasti v rámci které senzor měří

Hodnoty, které jsou větší než 256 jsou rozděleny na dva bajty a nižší bajt je uložen před vyšším. Kompletní přehled všech paramterů (i těch uživatelsky nepřístupných), jejich adres a stránek paměti je zobrazen v příloze B.

2.2.2 Volatilní parametry

Volatilní parametry odpovídají z větší části parametrům nevolatilním s tím rozdílem, že mezi volatilní parametry nepatří ethernetové parametry a parametry jako Default Mode a Offset, které se nastavují po startu senzoru.

2.3 Komunikační protokol

Komunikační protokol senzoru TLE1 je založen na bajtových příkazech. Příkazy pro senzor mohou být jak jednobajtové, tak i vícebajtové. Současně i odpověď od senzoru může mít různou délku.

2.3.1 Měřená data

Žádost o měřená data, tedy informace o vzdálenosti a výšce, má formát jednoho bajtu s hodnotou <**0x1X**>, kde X znamená množství požadovaných dat. Senzor odpoví 2^x měřenými daty. Pokud jsou tedy požadována jen jedna měřená data, bude příkaz pro senzor 0x10. Každá odpověď měřených dat má velikost 5 B a jejich formát je následující:

Přijatá data: <X VYŠŠÍ BAJT> <X NIŽŠÍ BAJT> <Y VYŠŠÍ BAJT> <Y NIŽŠÍ BAJT> <AUX BAJT>

Rozsah měřených dat X a Y je od 0 do 30000. Číslo představuje vzdálenost resp, výšku v mikrometrech. Formát posledního (AUX) bajtu je uveden v tabulce 2.1.

Tabulka 2.1: Položky AUX bajtu

bit	Název	Popis
7 (MSB)	OIN	Ojekt detekován v měřícím rozsahu
6	ZERO_CNT	Objekt je uvnitř procesovacího okna
5	OVER410_CNT	Signalizuje, že objekt překročil hodnotu 410 px uvnitř procesovacího okna
4	0	Rezervováno
3	USER_PAR_CHG	Uživatelské parametry se po zapnutí změnily
2...0	MODES	tříbajtové číslo reprezentující nastavený režim měření (viz. 2.1)

2.3.2 Obrazová data

Senzor umožňuje vyčtení celého obrazu zachycovaného pomocí CMOS čidla. Rozlišení tohoto obrazu je závislé na konkrétní nastavené hodnotě parametru Window – tedy rozsahu, ve kterém senzor měří. Zároveň je možné vyčíst obraz se čtvrtinovým rozlišením. Žádost o obrazová data má opět podobu jednoho bajtu, tentokrát s hodnotou <**0xD0**> pro plnou velikost a <**0xD1**> pro čtvrtinovou velikost.

Odpověď senzoru má podobu bajtového pole o velikost rozlišení nastaveného parametru Window. Položky pole představují jednotlivé pixely šedotónového obrazu.

Obraz je zasílán po řádcích bez ukončujícího znaku – programátor si tak sám musí udržovat informaci o rozlišení.

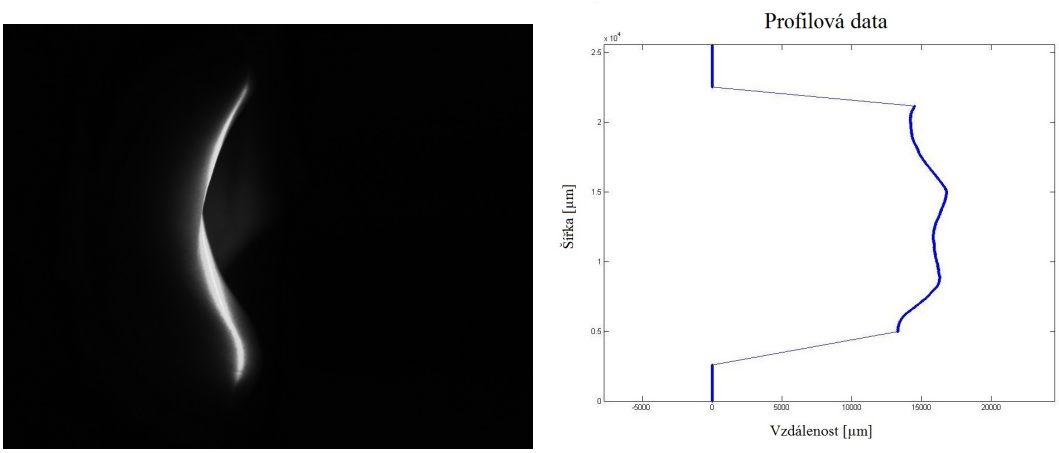
S obrazovými daty souvisí i vyčtení tzv. profilu. Tedy zpracovaných obrazových dat ve formě dvourozměrného pole se vzdáleností na ose X a výškou na ose Y. Velikost profilu je opět závislá na nastavení parametru Window. Pro přečtení těchto dat je nutné poslat příkaz <**0x60**>. Senzor odpoví zasláním bajtového pole, které má následující formát:

Přijatá data: <X VYŠŠÍ BAJT 1> <X NIŽŠÍ BAJT 1> <Y VYŠŠÍ BAJT 1> <Y NIŽŠÍ BAJT 1> ... <X VYŠŠÍ BAJT N> <X NIŽŠÍ BAJT N> <Y VYŠŠÍ BAJT N> <Y NIŽŠÍ BAJT N> <POŘADÍ VYŠŠÍ BAJT> <POŘADÍ NIŽŠÍ BAJT>

Počet bodů odpovídá nastavené výšce procesovacího okna Window. Na konci pole jsou ještě odeslány dva bajty obsahující pořadové číslo profilu. To je důležité zejména v režimu nepřetržitého přijmu profilových dat k identifikaci konkrétního profilu. Celková velikost pole tedy je:

$$\text{velikost} = (4\times \text{ výška procesovacího okna }) + 2$$

Na obrázku 4.9e je zobrazen příklad obrazových dat a k nim na obrázku 4.9f odpovídajcích dat proflových. Profilová data jsou převrácena kolem osy Y, protože jsou tak senzorem zpracovávána.



(a) Obrazová data

(b) Profilová data

Obrázek 2.3: Obrazová data a k nim odpovídající data profilová

2.3.3 Stream dat

Kromě běžného modelu žádost – odpověď umožňuje senzor i stream dat, tedy o nepřetržitý proud informací od vyslání příkazu k zahájení po odeslání příkazu pro ukončení. Výhoda streamu je hlavně v tom, že senzor data posílá maximální možnou rychlostí a žádná data nechybí. Pokud by byl například měřen rychle se pohybující objekt, mohlo by u běžného modelu žádost – odpověď dojít k tomu, že zrovna v datech, která budou chybět, bude důležitá informace Vyřízení žádosti senzoru chvíli trvá, zatímco u streamu dat nemusí senzor žádost pokaždé zpracovávat.

Senzor umožňuje streamovat dva typy dat – měřená data a profilová data. Pro zahájení streamu měřených slouží příkaz <**0x21**> a pro stream profilových dat pak příkaz <**0x22**>. Senzor po přijetí tohoto příkazu začne okamžitě odesílat data. Pro ukončení streamu je potřeba odeslat příkaz <**0x20**>. Na příkaz pro ukončení streamu senzor nijak neodpoví.

2.3.4 Práce s EEPROM pamětí

Data v EEPROM paměti jsou uložena v osmi různých stránkách, přičemž v každé jsou data uložena ve stejném pořádí (viz. 2.2.1). Při zápisu do této paměti je možné pracovat pouze s jednou stránkou zároveň. Senzor by uměl přepsat i s více stránkami zároveň, uživateli však tato možnost není z bezpečnostních důvodů umožněna. Příkaz pro uložení dat do EEPROM paměti má následující strukturu:

Odesílaná data: <0xB0> <ČÍSLO STRÁNKY> <OFFSET> <POČET BAJTŮ − 1> <BAJT 1> <BAJT 2> ... <BAJT N>

Senzor umožňuje najednou zapsat až 256 bajtů dat do stránek od 247 do 255. Data nejsou senzorem nijak ověřována. Je tak na programátorovi, aby zajistil, že do senzoru ukládá správná data. Senzor potvrdí provedení příkazu odesláním bajtu <POČET BAJTŮ - 1> z odeslané zprávy zpět uživateli.

Obdobný příkaz slouží i k přečtení údajů na jedné stránce EEPROM paměti. Má následující strukturu:

Odesílaná data: <0xA0> <ČÍSLO STRÁNKY> <OFFSET> <POČET BAJTŮ − 1>

Na tento příkaz senzor odpoví odesláním bajtového pole o jeden bajt většího než uživatel definoval ve čtvrtém bajtu odeslaného příkazu.

2.3.5 Nastavení volatilních parametrů senzoru

Nastavení volatilních parametrů má pokaždé stejnou strukturu. Při čtení uživatel odešle jeden bajt a jako odpověď dostane bajty dva (vyšší bajt první). Při zápisu uživatel odešle jeden bajt specifikující parametr a pak samotnou hodnotu ve dvou

bajtech (vyšší bajt první). Senzor na příkaz pro zápis v případě úspěšného provedení operace odpoví vrácením bajtu idenifikujícím parametr. Tabulka 2.2. uvádí příklad několika parametrů včetně příkazů pro jejich nastavení a přečtení. Ostatní parametry jsou kvůli většímu množství přesunuty do přílohy C. Parametry jsou pojmenovány tak, jak jsou uvedeny v manuálu Metralight (2012).

Tabulka 2.2: Volatilní parametry senzoru

Parametr	Popis	Čtení	Zápis	Rozsah
WINDOW	Průměrovací okno	0x82	8x8A	1 – 1023
THR	Prah průměrování	0x83	0x8B	1 – 254
TINT	Doba integrace	0x84	0x8C	1 – 1024
LSRPWR	Výkon laseru	0x85	0x8D	0 – 255

2.3.6 Ostatní příkazy senzoru

Kromě výše zmíněných příkazů existují i příkazy, které například umí vyčíst verzi firmwaru senzoru, vypnout laser nebo změnit aktivní režim měření. Příkaz pro vyčtení aktuální verze firmwaru senzoru je <**0xF0**>. Senzor odpoví dvěma bajty. Příkaz nastavující režim měření je <**0x3X**>, kde X je rovno módu, který chce uživatel nastavit. Standardní počet módu v senzoru je roven čtyřem, ale upravené verze senzoru mohou mít módů až osm. Senzor odpoví echo příkazem, kdy po úspěšném nastavení režimu vrátí doručenou zprávu zpět. Pokud je z nějakého důvodu nutné vypnout laser, stačí odeslat příkaz <**0x0x90**> a pro opětovné zapnutí <**0x91**>. Senzor opět odpoví echo příkazem.

2.3.7 Možné problémy při vytváření rozhraní

Komplikací pro vytváření rozhraní je hlavně možnost zákazníka do jisté míry ovlivnit chování senzoru. Zákazník má možnost zažádat o přidání určitých režimů měření dat či vlastních omezení parametrů. Pokud je například v systému zákazníka nemožné

používat laser na nejvyšší výkon, je toto omezení nastaveno již v senzoru. Zákazník má současně možnost zažádat o přidání dalších příkazů do komunikačního protokolu senzoru.

Firma Metralight zároveň již pracuje na výrobě senzoru s USB rozhraním. Je tak nutné softwarové rozhraní v tomto ohledu udělat dostatečně obecné, aby bylo možné v budoucnu přidávat další komunikační rozhraní.

3 Softwarové rozhraní

Jak bylo již naznačeno v poslední části předchozí kapitoly, rozhraní pro senzor TLE1 bude muset vyřešit několik překážek. Rozhraní musí být dostateěn flexibilní na to, aby pokrylo případné změny vlastností senzoru v budoucnu – ať už jde o změny komunikačního protokolu, režimů měření, vlastností parametrů či komunikačního rozhraní senzoru.

Druhým problémem, který je nutné vyřešit, je přiblížit senzor i programátorům, kteří nemají dlouholeté zkušenosti s programováním a nebyli by tak bez rozhraní schopni aplikaci napsat. Firma Metralight ani nemá zájem na to, aby zákazníci znali dopodrobna komunikační rozhraní senzoru. Vlastnosti senzoru se totiž mohou v čase měnit a bylo by pro firmu náročné, aby všechny zákazníky při každé změně informovala. V případě použitého rozhraní je zákazníkům buď odeslána nová verze knihovny, nebo jsou změny vyřešeny v rámci rozhraní tak, že zákazník nemusí ve své aplikaci nic změnit.

Rozhraní bude využíváno i v rámci společnosti Metralight pro jednodušší vytváření vlastních aplikací, které mohou například demonstrovat využití senzoru pro potřeby zákazníka.

Při vytváření softwarového rozhraní byly k dispozici pouze skripty pro Matlab. Ty do značné míry naznačily cestu, jakou by mělo být rozhraní vytvářeno, ale protože se programovací techniky jazyka C# a Matlabu značně liší, složily tyto skripty hlavně jako kontrola správnosti vytvářených funkcí. V rámci společnosti Metralight se pak pracuje i v jazyce VB .NET. Výhoda rozhraní vytvořeném v jazyce C# je ta, že je jej možné využít právě v jazyce VB .NET nebo načíst v rámci prostředí Matlab. Rozhraní se tak stane daleko univerzálnější a přístupnější pro větší množství programátorů.

3.1 Jmenná konvence

V rámci všech zdrojových kódů je použitá stejná jmenná konvence, která vychází z kníhy Sharp (2010). Pro psaní víceslovných bázvů je s výjimkou konstant použit styl psaní zvaný CamelCase – tedy že každé slovo začíná okamžitě za druhým, ale s velkým písmenem. Názvy jmenných prostorů, tříd, enumerátorů a souborů začínají vždy velkým písmenem. Názvy globálních proměnných mají prefix ,,m_". Pokud funkce obsahuje parametry, ty mají prefix ,,_". Názvy lokálních proměnných a funkcí pak vždy začínají malým písmenem.

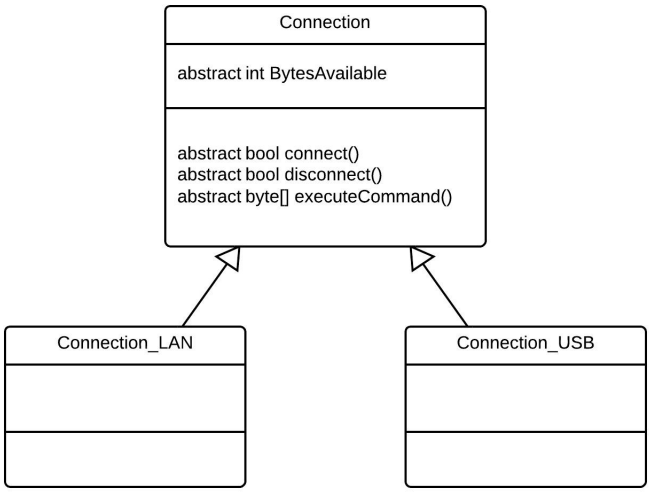
3.2 Komunikační rozhraní

V současné době je senzor vybaven pouze ethernetovým rozhraním. Do budoucna se však počítá s další verzí, která bude vybavena také USB rozhraním. Je proto nutné přizpůsobit rozhraní tak, aby bylo dostatečně obecné a aby přidáním nového způsobu připojení nemusela být měněna celá struktura rozhraní.

Pro tyto účely je možné využít programátorského prostředku – rozhraní, v něm definovat potřebné metody jako například metodu pro připojení a odpojení a ve třídách definujících jednotlivá rozhraní toto rozhraní implementovat. Nevýhoda rozhraní je ale v tom, že není možné definovat tělo metod definovaných právě uvnitř rozhraní. To je nevýhodou zejména v případě, kdy by bylo vhodné sjednotit metody vypisující řetězce či metody ošetřující chyby. Jazyk C# pro tyto účely nabízí abstraktní třídy. Ty mají mnoho společných vlastností právě s rozhraním. Pokud se metody uvnitř abstraktní třídy definují jako abstraktní, je programátor, stejně jako v případě rozhraní, nucen metody implementovat případně předefinovat. Výhodou abstraktní třídy je ale právě to, že je možné definovat těla metod.

Protože ale mohou třídy v jazyce C# implementovat rozhraní a zároveň dědit od abstraktních tříd, bylo by možné použít kombinaci obojího. Při využití abstraktních metod abstraktních tříd ale není nutné rozhraní vůbec použít. Na obrázku 3.1 je

naznačena koncepce abstraktní třídy pro připojení k senzoru a zdrojový kód 3.1 pak ukazuje konkrétní použití.



Obrázek 3.1: UML diagram tříd pro připojení

Zdrojový kód 3.1: Abstraktní třída pro komunikační rozhraní

```
1  using System;

3  namespace TLESensorClass.Connections
   {
5      public enum Status : int { Online, Occupied, Offline };

7      public abstract class Connection
       {
9          public event EventHandler TimeoutError;

11         public abstract int BytesAvailable { get; }

13         protected void OnTimeoutError(EventArgs e)
           {
15             EventHandler handler = TimeoutError;
               if (handler != null)
17             {
                   handler(this, e);
19             }
           }
21
           public abstract Status testConnection();
```

```
23          public abstract bool connect();
25          public abstract bool disconnect();
27          public abstract byte[] executeCommand(byte _command,
                byte[] _data, int _respondSize);
29      }
    }
```

Zdrojový kód 3.1 zobrazuje abstraktní třídu **Connection**, která definuje abstraktní metody connect(), disconnect(), executeCommand() a testConnection(). Tyto metody musí každá třída, která od třídy Connection dědí, překrýt. Tedy použít klíčové slovo override a díky němu definovat vlastní implementaci zmíněných funkcí.

Význam metod connect() a disconnect() je zřejmý. Metoda testConnection() slouží k testu připravenosti nebo dostupnosti senzoru. Jako výsledek vrací jeden z prvků enumerátoru Status. Metoda executeCommand() pak slouží k provedení příkazu pro senzor. Jako parametr přijímá příkaz a případně i jeho data. Zároveň je zde uvedeno, kolik bajtů se má ze senzoru přečíst a tato jsou následně vrácena v podobě bajtového pole. Událost TimeoutError definuje akci, která se má stát v případě nedoručení dat v zadaném čase. V tomto případě handler pouze předá událost dál.

3.3 Implementace funkcí senzoru

Aby měl uživatel možnost se senzorem pracovat, bylo nutné implementovat všechny funkce, které senzor poskytuje – tedy přístup k měřeným datům, obrazovým datům, parametrům a funkcím nastavujícím chování senzoru.

3.3.1 Komunikační protokol

Pro usnadnění práce s komunikačním protokolem a zlepšení přehlednosti celého kódu byly všechny příkazy zapsány ve formě enumerátoru. Enumátor je bajtový, takže přímo nahrazuje příkazy. Konkrétní hodnoty příkazů je možné zaspat jak dekadicky, tak šestnáctkově. Konkrétní příklad převodu komunikačního rozhraní je uveden ve zdrojovém kódu 3.2. S jednotlivými příkazy v kódu se tak již nadále nepracuje jako s čísly, ale pomocí jejich identifikátoru.

Zdrojový kód 3.2: Komunikační rozhraní

```
using System;

namespace TLESensorClass.Sensor
{
    public enum CommandProtocol : byte
    {
        Get_Profile = 0x60,
        Get_StartLine_limits = 0x80,
        Get_NumOfLines_limits = 0x81,
        Get_AverageWindow = 0x82,
        Get_Threshold = 0x83,
        Get_TimeOfIntegration = 0x84,
        Get_LaserPower = 0x85,
        Get_LineProcMode = 0x86,
        Get_PeakSize = 0x87
    };
}
```

3.3.2 Zpracování měřených dat

Přístup ke konkrétním měřeným datům je bez odpovídajícího rozhraní poměrně komplikovaný. Pro získání konkrétních čísel je nutné znát informace o tom, jak jsou řazeny bajty a jaké údaje konkrétně obsahují. Jak bylo popsáno v kapitole 2.3.1, senzor na žádost o měřená data odpoví právě pěti bajty. První dva bajty jsou hodnota vzdálenosti, druhé dva bajty hodnota výšky a poslední bajt je vyhrazen pro

dodatečné informace. Z dodatečných informací se dále v programu používají údaje o tom, zda je objekt v měřené oblasti (současně indikováno diodou přímo na těle senzoru) a o aktuálně aktivním módu měření. Pro účely převodu bajtového pole na konkrétní informace byla vytvořena třída **SensorDataFormat**. Tato třída přijímá jako parametr konstruktoru právě bajtové pole a nabízí pak přístup ke konkrétním hodnotám přes veřejné položky třídy. Pro převod informací z posledního bajtu na dodatečné informace je využita třída **BitArray**, která poskytuje funkce pro přístup k jednotlivým bitům zadaného bajtu. Příklad převodu informace o detekovaném objektu a módu je ukázán ve zdrojovém kódu 3.3.

Zdrojový kód 3.3: Zpracování dodatečných informací

```
1  public SensorDataFormat(byte[] _data)
   {
3      byte auxByte = _data[4];

5      if (auxByte > 0)
       {
7          BitArray bitArray = new BitArray(auxByte);

9          mode = (byte)(auxByte & 0x07);

11         try { oin = bitArray.Get(7); }
           catch (ArgumentOutOfRangeException) { oin = false; }
13     }
   }
```

Výsledná třída kromě informací uvedených ve zdrojovém kódu 3.3 obsahuje i funkce pro získání ostatních údajů a jsou zde ošetřeny a předány dál všechny vzniklé výjimky a chyby.

Všechny měřené hodnoty je zároveň možné ukládat do historie pro pozdější zpracování určitého množství po sobě jdoucích hodnot. Ukládání hodnot má na starosti třída **ValueHistory**. Ta vytvoří frontu o velikosti, kterou zadá uživatel při jejím vytváření. Protože je však nutné, aby třída zpracovávala údaje velmi rychle (řádově stovky údajů za sekundu), je potřeba použít vhodnou datovou strukturu. Jazyk C#

pro tyto účely poskytuje třídu **Queue**. Ta obsahuje metody Enqueue() a Dequeue(). Rychlost ukládání a mazání údajů je násobně rychlejší než při použití klasického pole nebo Listu. Současně se fronta sama stará o to, aby udržovala pouze určité množství hodnot.

3.3.3 Práce s EEPROM pamětí

V paměti EEPROM je uloženo velké množství údajů. Bylo proto potřeba vytvořit třídu, která tyto údaje bude zpřehlední. Jak bylo popsáno v kapitole 2.2.1, jsou v EEPROM uloženy údaje dvojího typu. Jsou zde jak parametry senzoru, tak parametry nastavující ethernetové připojení. Z tohoto důvodu byly vytvořeny třídy **EEPROMParameters** a **EEPROMEthernet**. Tyto dvě třídy obsahují metody pro převod bajtového pole získaného ze senzoru na jednotlivé informace a zpět – tedy z konkrétních údajů na bajtové pole. Tyto metody je však možné použít jen při vyčítání kompletní sady parametrů, kdy je přečtena celá stránka paměti. Zdrojový kód 3.4 uvádí jeden z konstruktorů třídy EEPROMEthernet, který přijímá jednotlivé parametry a vytváří z nich bajtové pole. Pro urychlení tohoto procesu je použita třídy **MemoryStream**. Ta předchází nutnosti vytvářet nové bajtové pole o předem dané velikosti.

Zdrojový kód 3.4: Zpracování EEPROM dat

```
public EEPROMEthernet(IPAddress _address, IPAddress
    _subnetMask, IPAddress _gateway)
{
    m_IPaddress = _address;
    m_subnetMask = _subnetMask;
    m_gateway = _gateway;

    MemoryStream ms = new MemoryStream(new byte[DATA_SIZE],
        0, DATA_SIZE, true, true);
    ms.Write(_address.GetAddressBytes(), 0, 4);
    ms.Write(_subnetMask.GetAddressBytes(), 0, 4);
    ms.Write(_gateway.GetAddressBytes(), 0, 4);

    m_eepromEthernet_field = ms.GetBuffer();
}
```

3.3.4 Zpracování profilových dat

Jak bylo popsáno v kapitole 2.3.2, profilová data představují předzpracovaný obraz ve formě dvourozměrného pole. Aby bylo možné převést strukturu, kdy jsou jednotlivé body posílány za sebou, do podoby dvou polí nebo jednoho pole dvourozměrného, bylo potřeba vytvořit speciální funkci. Ta se jmenuje **ProfileData** a v konstruktoru přijímá bajtové pole dané velikosti. Toto pole je následně rozděleno na dvě pole, přičemž v jednom poli jsou hodnoty vzdálenosti (X) a ve druhém hodnoty výšky (Y). Pokud bude v budoucnu potřeba rozšířit tuto strukturu o dvourozměrné pole nebo jenom jinou definici jednorozměrného pole, stačí pouze aktualizovat právě tuto třídu.

3.3.5 Práce s parametry

Aby byla sjednocena i práce s parametry, byla vytvořena třída **ActualSettings**. Tato třída poskytuje přístup k proměnným, které se mají v budoucnu použít, anebo se používají za krátkou dobu vícekrát. Jedná se o jednoduchou strukturu složenou ze proměnných, které mají veřejné metody gettery a settery. Díky tomu je možné do instance této třídy uložit hodnotu a tu později číst.

Možnost jednotného vyčtení všech parametrů sice byla také diskutována, ale parametrů je tolik, že by to zabralo delší dobu a málokdy je potřeba parametry využít opravdu všechny. Pokud by však taková situace v budoucnu nastala, jednoduchou modifikací třídy ActualSettings lze dosáhnout i tohoto chování.

Zvláštním případem parametrů jsou parametry ROI a Window. Tyto parametry se totiž skládají z více dílčích parametrů. Oba mají počáteční rádek, počet řádků, počáteční sloupec a počet sloupců. Jedná se tak v podstatě o obdélníky se zadaným levým horním rohem, šířkou a výškou. Aby nebylo nutné pokaždé přemýšlet, který parametr je vlastně potřeba a jaký parametr se má načíst, byla vytvořena pomocná třída **PointSize**. Tato třída v sobě zapouzdřuje všechny vlastnosti obdélníku a pracuje se všemi parametry zároveň. Jako konkrétní příklad je možné uvést situaci,

kdy progamátor potřebuje znát šířku okna Window – buď může použít parametr WINDOW_NUM_OF_COLUMNS anebo intuitivně použít vlastnost window.Height instance třídy PointSize.

3.4 Třída SensorClass

Třída SensorClass v sobě zapouzdřuje všechnu funkcionalitu zmíněnou v předchozích odstavcích a je to právě tato třída, se kterou uživatel pracuje. Po vytvoření instance umožňuje třída se k senzoru připojit, číst a zapisovat do něj data.

Na následujících řádcích budou popsány důležité položky a funkce třídy SensorClass spolu s jejich stručným popisem.

public SensorClass(Connection _connection)

Konstruktor třídy, který přijímá jako parametr instanci třídy Connection – ta definuje parametry připojení k senzoru a uloží ji jako neveřejnou proměnnou třídy SensorClass.

public bool connect()

Připojí se k senzoru. V případě, že se připojení povede, vrací true. V opačném případě vrací false. K otestování dostupnosti senzoru lze využít funkci testConnection() poskytovanou třídou Connection.

public bool disconnect()

Odpojí se od senzoru. Instance řídy Connection ale zůstane ve třídě uložena a je možné se k senzoru později zase připojit bez nutnosti znovu zadávat údaje pro připojení.

public byte[] sensorUnlock()

Senzoru je odeslána speciální sekvence příkazů, které umožňují práci s EEPROM pamětí či vyčítání obrazu. Odemčení je jakousi bezpečnostní pojistkou pro situace, kdy v programu nastane chyba a program by začal senzoru posílat nesmyslná

data – nedojde tak k náhodnému přepsání paměti EEPROM. V případě úspěšného odemčení odešle senzor zpět první zaslaný bajt. Funkce je vypsána ve zdrojovém kódu 3.6. Funkce public byte[] sensorUnlock2() funguje obdobně jako funkce předchozí a liší se pouze v hodnotách příkazů. Bez zadání tohoto příkazu není možné smazat obsah celé paměti EEPROM naráz – musí se mazat po jednotlivých stránkách.

Zdrojový kód 3.5: Odemčení senzoru

```
1 public byte[] sensorUnlock()
  {
3     byte[] recievedData = new byte[1];
      recievedData = m_connection.ExecuteCommand(0, new byte[]
          { 254, 241, 249, 224 }, 1);
5
      return recievedData;
7 }
```

public byte[] sendCommand(byte[] _command, int _respondSize)

Tato funkce odešle senzoru příkaz specifikovaný parametrem _command a vyčká na doručení požadovaného množství dat. Ta jsou pak vrácena jako bajtové pole. Funkce je současně i přetížena a umožňuje vynechat parametr _respondSize.

Zdrojový kód 3.6: Odeslání příkazu senzoru

```
1 public byte[] sendCommand(byte[] _commandData, int
      _respondSize)
  {
3     return m_connection.ExecuteCommand(0, _commandData,
          _respondSize);
  }
```

public byte[] readData(int _respondSize)

Tato funkce pouze přečte data aniž by nějaká data odeslala. To je výhodné zejména v případě, kdy chceme vyčíst všechna data, která má senzor v bufferu.

public int BytesAvailable

Proměnná udržující v sobě informaci o množství bajtů, které jsou v současné době v bufferu senzoru.

public SensorDataFormat get1Data()

Přečte právě jedna měřená data a předá je ve formátu definovaném třídou SensorDataFormat (viz. kapitola 3.3.2). Po přijetí měřených dat je zároveň aktivována událost onMeasuredDataRecieve, která umožňuje data číst nezávisle na měřené hodnotě. Tohoto se využívá například při kontinuálním čtení měření dat, kdy data nejsou vracena konkrétní metodě jako návratová hodnota, ale jsou čtena ve funkci upravující reakci na událost.

Zdrojový kód 3.7: Čtení měřených dat

```
public SensorDataFormat get1Data()
{
    byte[] recievedData = new byte[5];
    recievedData = m_connection.ExecuteCommand((byte)
        CommandProtocol.Get_1Data, null, 5);

    SensorDataFormat sensorData = new SensorDataFormat(
        recievedData);

    if (onMeasuredDataRecieve != null) onMeasuredDataRecieve(
        sensorData);

    return sensorData;
}
```

Pro přečtení většího množství měřených dat je připravena funkce public byte getNData(int _data, CommandProtocol _com), která umožňuje vyčíst počet dat definovaný v kapitole 2.3.1.

public byte[] getImage()

Funkce umožňuje přečíst ze senzoru jeden údaj obrazových dat (viz. kapitola 2.3.2). Na začátku je zkontrolována šířka a výška parametru Window, která určuje velikost

obrazu a podle ní je určena výsledná velikost. Pro získání obrazu se čtvrtinovou velikostí je připravena funkce public byte[] getImageReduced(). Při přijetí jakéhokoliv typu obrazových dat je aktivována událost OnImageRecieve. Ta obdobně jako událost u měřených dat informuje o přijetí nového obrazu bez nutnosti číst data z návratové hodnoty.

public ProfileData getProfile()

Funkce pracuje obdobně jako funkce předchozí, ale tentokrát vyčítá data o profilových datech. Opět je na začátku nutné ověřit Windows. Tentokrát ale stačí informace o výšce. Pokud programátor z nějakého důvodu nechce data ve formě třídy ProfileData (viz. kapitola 3.3.4), může použít funkci public byte[] getProfileBytes().

public EEPROMParameters getEEPROMParameters(byte _pageNum)

Tato funkce přečte stránku paměti, která je určena parametrem _pageNum. Údaje jsou vráceny jako instance třídy EEPROMParameters (viz. kapitola 3.3.3). Ve funkci není kontrolována validita hodnoty _pageNum, protože může v budoucnu dojít k přidání nových stránek a obecnost této funkce by ztratila na významu. Podobnou funkcí, ale pro ethernetová data, je funkce public EEPROMEthernet getEEPROMEthernet().

public byte[] setEEPROMParameters(EEPROMParameters _eepromData, byte _pageNum)

Následující funkce slouží naopak pro uložení EEPROM dat do zadané stránky EEPROM paměti. Konzistence dat je ověřena již při vytváření instance třídy EEPROMParameters, která je předaná jako parametr. Opět existuje stejná funkce i pro ethernetová data – public byte[] setEEPROMEthernet(EEPROMEthernet _eepromData).

private int getValue(CommandProtocol _com)

Tato funkce je předpisem dalších funkcí, které čtou parametry senzoru (viz. zdrojový kód 3.8). Aby nebylo nutné pro každý parametr vytvářet samostatnou funkci, jsou

funkce derivovány právě z této tím, že se předá název příkazu, čtoucí konkrétní parametr, jako parametr funkce. Pokud je z nějakého důvodu potřeba přečíst parametr jako pole bajtů, je možné využít funkci private byte[] getValueBytes(CommandProtocol _com).

Zdrojový kód 3.8: Šablona funkce pro čtení parametrů

```
1  private int getValue(CommandProtocol _com)
   {
3      byte[] recievedData = new byte[2];
       recievedData = m_connection.ExecuteCommand((byte)_com,
          null, 2);
5      int result = recievedData[0] * 256 + recievedData[1];

7      return result;
   }
```

Ukázka čtení konkrétního parametru je pak ve zdrojovém kódu 3.9.

Zdrojový kód 3.9: Čtení parametru

```
   public int getThreshold()
2  {
       return getValue(CommandProtocol.Get_Threshold);
4  }
```

private byte[] setValue(CommandProtocol _com, int _value)

Princip definování předpisu funkce a jejím následným použití u parametrů je použit i při ukládání. Jako parametr funkce je nutné uvést příkaz, která konkrétní parametr ukládá a jeho vlastní hodnota jako datový typ int. Funkce hodnotu převede na dva bajty a ty otočí, protože vnitřní funkce jazyka C# ve výchozím nastavení pracuje s pořadím, kdy je spodní bajt první, v senzoru je to ale naopak.

Zdrojový kód 3.10: Šablona funkce pro uložení parametrů

```
private byte[] setValue(CommandProtocol _com, int _value)
{
    byte[] valueBytes = BitConverter.GetBytes(_value);
    Array.Reverse(valueBytes, 0, 2);

    byte[] recievedData = new byte[1];
    recievedData = m_connection.ExecuteCommand((byte)_com,
        valueBytes, 1);
    return recievedData;
}
```

3.4.1 Ostatní funkce třídy SensorClass

Třída SensorClass dále obsahuje jednodušší funkce k provedení základních operací. Mezi ně patří například funkce pro započetí a ukončení kontinuálního čtení měřených a profilových dat či funkce pro nastavení aktuálního režimu měření dat a používané stránky paměti EEPROM. Třída také obsahuje funkci public string getFWVersion(), která ze senzoru vyčte informaci o verzi firmwaru. Informace o verzi má ale předem danou strukturu, na kterou je nutné získanou hodnotu převést a vrátit jako řetězec.

3.5 Výsledná struktura rozhraní

Rozhraní, které bylo pojmenováno jako TLESensorClass tedy obsahuje následující třídy:

- Connections
  - Connection
  - Connection_LAN
  - Connection_USB
- Sensor
  - ActualSettings
  - CommandProtocol

- EEPROMEthernet
- EEPROMParameter
- PointSize
- ProfileData
- **SensorClass**
- SensorDataFormat
- ValueHistory

4 Realizace ukázkové aplikace

Aby bylo možné ověřit schopnosti navrženého softwarového rozhraní v praxi, bylo nutné ho využít v rámci konkrétní aplikace. Aplikace byly nakonec vytvořeny dvě. První aplikace má sloužit hlavně jako demonstrace všech vlastností a schopností senzoru a s jejím využitím se počítá v rámci firmy Metralight. Aplikace byla nazvána TLE Studio.

Druhá aplikace, nazvaná Profile Scanner, vznikla za účelem demonstrace využití senzoru v rámci Technické univerzity v Liberci. Tato aplikace ukazuje použití senzoru při odměřování různých objektů jako jsou například desky plošných spojů, šrouby či kovové profily.

4.1 TLE Studio

Jak již bylo zmíněno výše, hlavním účelem aplikace je ukázat všechny schopnosti a vlastnosti senzoru TLE1. Bylo tak nutné vyřešit několik základních problémů. Prvním problémem je uživatelské rozhraní. Musí být natolik jednoduché, aby jej mohli ovládat i lidé, kteří nejsou programátoři a zároveň natolik komplexní, aby obsáhlo všechny funkce senzoru. Aplikace musí být současně natolik robustní, aby neskončila chybou například v případě, že uživatel zadá špatné údaje do polí pro vstup textu – bylo tedy zásadní, aby byly ošetřeny všechny chybové stavy.

Druhým problémem bylo navržení vhodné reprezentace měřených dat. U měřené hodnoty je situace jasná – jedná se pouze o jedno číslo. U profilových dat a dat obrazových je ale situace jiná, protože data lze vizualizovat různými způsoby.

4.1.1 Uživatelské rozhraní

Uživatelské rozhraní je základní a nejdůležitější částí aplikace. Právě skrze něj uživatel komunikuje se senzorem a díky němu může nastavovat údaje nebo naopak jiná data číst. První návrhy byly realizovány pomocí C# frameworku Windows Forms. Protože ale bylo nutné s uživatelským rozhraním pracovat více detailně, bylo rozhodnuto o přechodu na jiný framework – konkrétně framework WPF. Ten umožňuje například jednodušší změnu barev, grafické efekty nebo stínování. Jeho výhodou je současně i to, že uživatelské rozhraní definuje v XML souboru. Je tak oddělena funkční část aplikace od grafického návrhu. Microsoft pro jednodušší práci s WPF dodává i samostatnou aplikaci Blend.

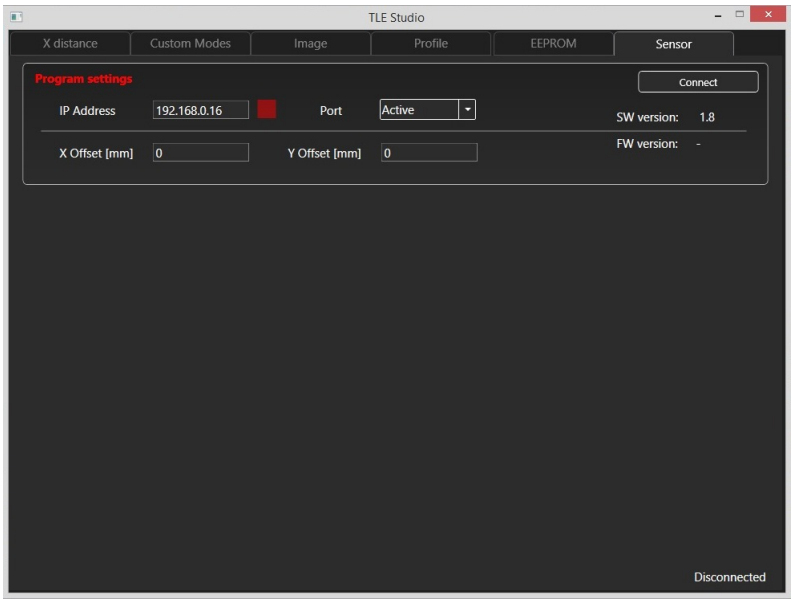
První vlastností uživatelského rozhraní aplikace TLE Stuido, která byla řešena, je velikost okna. Protože je možné, že se aplikace bude spouštět i na průmyslových počítačích, bylo by nevhodné použít velké okno. Bylo proto rozhodnuto, že maximální velikost okna bude 1024 x 768 pixelů. Aplikace je však ještě o něco menší, protože se musí počítat i s velikostí nabídky Start operačního systému Windows.

Druhou důležitou vlastností je barva okna a celkové barevné schéma aplikace. První návrhy měly bílé pozadí a černý text. To se však později ukázalo jako nevhodné, protože delší sledování obrazovky monitoru s bílým pozadím velmi namáhá oči. Výsledné schéma je tedy tmavé se světlým textem. Výhodou je současně i to, že by tmavé barvě více vyniknou další barvy jako červená nebo zelená. Současně se změnou barevného schématu bylo ale nutné znovu navrhnout grafickou úpravu všech ovládacích prvků, protože výchozí nastavení WPF je černý text na bílém pozadí. Výsledné grafické schéma je zobrazeno na obrázku 4.1.

Po grafické stránce aplikace bylo nutné navrhnout i její logickou strukturu, tedy oddělit například měřená data od dat obrazových. Pro tyto účely byla aplikace rozdělena do několika záložek:

- **X Distance** – záložka pro základní čtení měřených hodnot ve výchozím režimu
- **Custom Modes** – záložka pro čtení měřených hodnot v další režimech

- **Image** – záložka pro zobrazování obrazových dat a nastavení souvisejících parametrů
- **Profile** – záložka pro čtení profilových dat
- **EEPROM** – záložka pro práci s pamětí
- **Sensor** – záložka pro připojení k senzoru a nastavení dočasných parametrů



Obrázek 4.1: Okno aplikace TLE Studio

Protože se v aplikaci předpokládá velké množství údajů zadávaných uživatelem, bylo nutné vytvořit jednotný způsob validace těchto dat. Pro tyto účely byla standardní komponenta TextBox, která slouží k zadávání uživatelského vstupu, rozšířena o nové funkce. Nově tedy pole umožňuje zadat typ hodnot, které do něj budou vkládány a na základě tohoto údaje ověřit, zda jsou data správná. Pole podporuje následující vstupy:

- **NumberDecimal** – celočíselný vstup
- **NumberFloat** – vstup čísel s plovoucí řádovou čárkou
- **Boolean** – hodnoty true a false (nebo i 1 a 0)
- **IP String** – řetězec specifikující adresu IP
- **Text** – klasický řetězec

Pro validaci pak stačí použít funkci checkValue(), která hodnoty ověří. V případě, že jsou údaje zadány chybně, je uživatel na tuto skutečnost upozorněn zobrazením bílého křížku vedle pole a obarvením hodnoty na červeno. V případě korektního zadání hodnot se grafická úprava pole nemění. Třída upraveného pole TextBox byla nazvána **TextBox_Improved**.

Důležitou funkcí aplikace je časovač. Ten se stará o to, aby byly v pravidelných intervalech čteny údaje o měřené hodnotě, obrazu a profilu a to v závislosti na právě aktivované záložce. Při přepnutí do záložek Sensor nebo EEPROM je časovač zastaven, protože v těchto záložkách není nutné periodicky číst hodnoty.

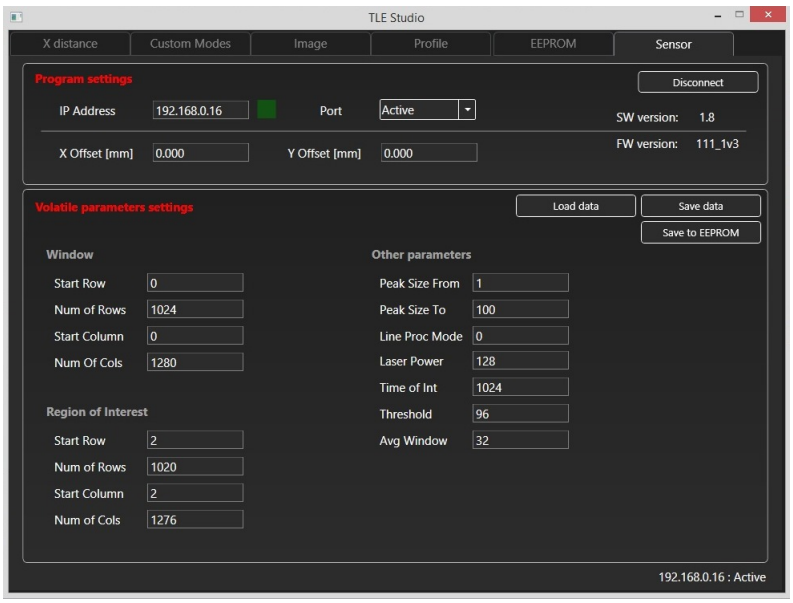
4.1.2 Záložka Sensor

Záložka Sensor je výchozí záložkou, která se uživateli aktivuje po spuštění aplikace. Bez připojení k senzoru není možné tuto záložku opustit a současně nejsou bez připojení zobrazeny všechny ovládací prvky. Stránka obsahuje základní prvky pro připojení k senzoru - IP adresu a port. Senzor umožňuje připojení ke dvěma portům – 1024 a 1028. Protože by ale bylo pro uživatele zbytečné si porty pamatovat, byly nahrazeny slovním ekvivalentem Active a Pasive. Po připojení k portu Active je uživateli dovoleno pracovat se všemi funkcemi senzoru. Připojí-li se ale uživatel k portu Pasive, nemůže v senzoru nic nastavovat ale pouze číst. Dalšími prvky na stránce jsou informace o verzi firmwaru, verzi softwaru a pole pro nastavení offsetu hodnotu X a Y. Do nich je možné zadat jak kladná, tak záporná čísla a bude použit v rámci celé aplikace. Všechny doposud zmíněné prvky jsou sjednoceny do panelu s názvem Program settings.

Dalším panelem na stránce, který se ale zobrazí až po připojení k senzoru, je panel Volatile parameter settings, který vypisuje všechny údaje uložené v dočasné paměti senzoru. Zde je možné hodnoty jak číst, tak měnit a zapisovat do paměti. Je zde i tlačítko pro zkopírování všech parametrů do EEPROM paměti. Po jeho stisknutí se zobrazí nové okno, které uživatele vyzve k zadání stránky paměti, do které mají být hodnoty uloženy. Všechna vstupní pole jsou typu TextBox_Improved.

Před uložením jsou tedy hodnoty ověřeny a v případě chybného zadání není uložení umožněno. Po připojení k senzoru je v dolní části okna zobrazena informace o IP adrese senzoru a konkrétním portu. Tato informace je viditelná při aktivaci jakékoliv záložky.

Podoba záložky v případě nepřipojení k senzoru je na obrázku 4.1, připojenou záložku pak zobrazuje obrázek 4.2.



Obrázek 4.2: Záložka Sensor aplikace TLE Studio

4.1.3 Záložka X Distance

Hlavním účelem této záložky je poskytnout uživateli informaci o měřené hodnotě v režimu měření MODE_0 (viz. 2.1). V horní části okna je tak zobrazena velkým písmem samotná hodnota s jednotkou spolu s maximem a minimem. Tyto dva údaje lze resetovat kliknutím na jakékoliv číslo či stisknutím klávesy R. Ve spodní části okna pak nabíhá graf reprezentující historii s časem na ose X a hodnotami na ose Y. Graf je vyplněn červenou barvou s mírnou průhledností, aby byly stále čitelné popisky os. Obrázek 4.3 zobrazuje podobu záložky.

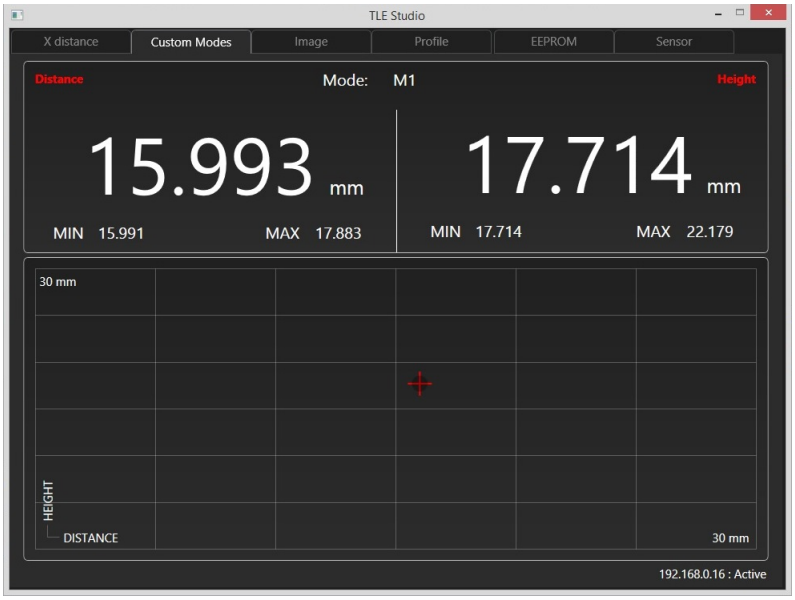
Při zobrazování měřené hodnoty nastal problém s použitím tečky nebo čárky jako desetinného oddělovače. Jazyk C# se ve výchozím nastavení přizpůsobí jazyku systému. Tato vlastnost má ale za následek nepředvídatelné chování na anglické verzi systému Windows, protože se nastavení projevuje například i při převádění hodnot s řetězců na čísla. Bylo proto rozhodnuto, že se v programu napevno nastaví tečka jako desetinný oddělovač. Program se tak chová stejně na při použití jakékoliv jazykové mutace systému Windows.

Obrázek 4.3: Záložka X Distance aplikace TLE Studio

4.1.4 Záložka Custom Modes

Tato záložka slouží k zobrazování měřených dat v jiných režimech než je režim MODE_0. Kromě vzdálenosti tedy zobrazuje i údaj o výšce. Oba údaje jsou zároveň doplněny svojí maximální a minimální hodnotu, kterou lze opět resetovat stiskem klávesy R nebo klepnutím na jakékoliv pole. Ve výchozím nastavení senzoru jsou dostupné režimy měření MODE_1 až MODE_3. K přepnutí režimu stačí kliknout na název režimu v horní části obrazovky.

Pro zlepšení přehlednosti a větší názornost toho, co je vlastně měřeno, byl ve spodní části záložky vytvořen graf se vzdáleností na ose X a výškou na ose Y. V tomto grafu je zakreslen bod korespondující s aktuálně měřenou hodnotou. Aby byly trošku naznačeny i minulé hodnoty, má tento bod ,,ocas", který je právě na základě předchozích hodnot vykreslován. Podoba záložky je na obrázku 4.4,



Obrázek 4.4: Záložka Custom aplikace TLE Studio

4.1.5 Záložka Image

Záložka Image slouží k zobrazení obrazových dat senzoru. Záložka je rozdělena na několik částí. V levé části je zobrazen samotný obraz, pod ním pak graf řezu označeného řádku a pravá část je vyhrazena pro nastavení parametrů.

Funkce vykreslování obrazu se skládá z několika částí. První částí je převod bajtového pole na bitmapu, která je pak vykreslována. Jazyk C# ale bohužel nemá vestavěnou funkci, která tento problém přímo řeší. Bylo proto nutné napsat funkci, která tento převod zajistí. Protože jsou ale obrazy vykreslovány rychle za sebou, bylo nutné při vytváření myslet i na rychlost a optimalizaci. Pro tyto účely bylo přikročeno k využití ukazatelů. Ty pracují v tzv. ,,unsafe"bloku kódu, který není při

kompilaci striktně kontrolován. To má však za nevýhodu to, že může snadno dojít k přespání paměti, která přepsána býti nemá. Microsoft sám upozorňuje na únik paměti, který u této funkce vzniká. Bylo proto nutné nenechávat úklid paměti plně na garbage collectoru, ale mazat paměť manuálně. Výsledkem je funkce, která vykreslí snímek při rozlišení 640 x 512 pixelů za méně než 10 ms. Druhým problémem při vykreslování obrazu je to, že obraz může mít různou velikost v závislosti na nastavení parametru Window. Je tedy nutné obraz po převodu na bitmapu přesunout na správnou pozici.

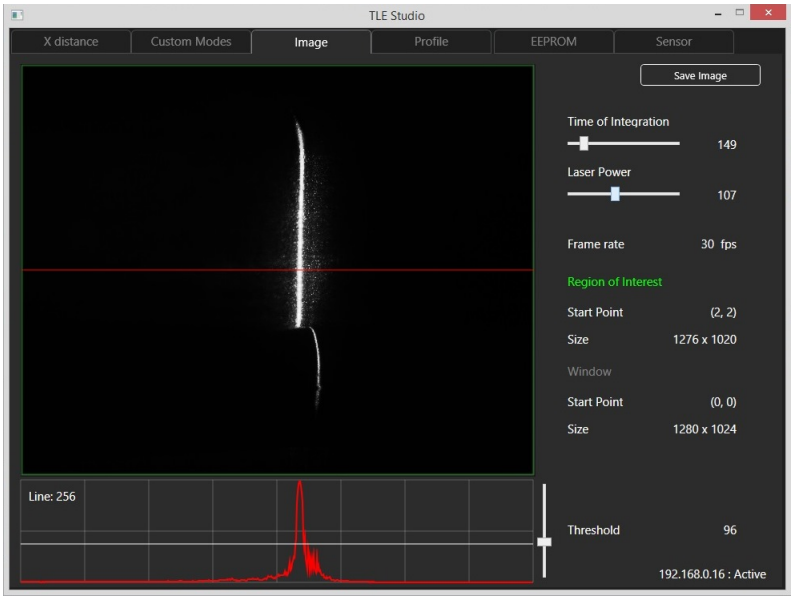
Další funkcí, kterou má záložka Image na starosti je nastavování velikostí oblastí Window a Region of Interest. Jak bylo popsáno v kapitole 3.3.5 jedná se o obdélníky specifikující aktivní oblast, kde senzor měří resp. zpracovává data. Pro jejich nastavení existuje několik možností. Nejjednodušší možností by bylo použít klasická pole a šířku, výšku a počáteční bod do nich zapsat. Toto by ale uživateli znemožnilo vizuálně zkontrolovat správnost zadaných údajů. Další možností proto je použití posuvníků vedle obrazu. Tato komponenty umožňuje pomocí jezdce nastavit hodnotu a jeden posuvník může mit jezdců vice. Nebyl by tak problém nastavit oblast s použitím dvou posuvníků. Toto řešení ale nevypadá vizuálně hezky a ergonomie ovládání není ideální. Bylo proto přikročeno k vytvoření komplexní grafické komponenty, která vykreslí oblasti Window a ROI přímo do obrazu a jejich změnu umožňuje tažením za okraje obdélníku. Uživatel tak má okamžitou vizuální zpětnou vazbu o tom, jak parametry nastavil.

S obrazem souvisí i graf řezu aktivního řádku. Tento graf je umístěn přímo pod obrazem a po kliknutí na libovolný řádek ho vykreslí jako graf s pixely na ose X a jejich hodnotou na ose Y. Tento graf slouží primárně pro jednodušší nastavení parametru Threshold – tedy hodnoty, která specifikuje minimální hodnotu pro zpracování dat. Threshold je v grafu vykreslen jako bílá čára a vizuálně uživatele informuje o svojí hodnotě. Tu je možné měnit pomocí posuvníku napravo od grafu.

Zobrazení obrazových dat i nastavení oblastí bylo spojeno do jedné vizuální komponenty, která má několik vrstev. Ve spodní vrstvě se vykresluje samotný obraz. Ten

je pozicován v rámci černého okna. Nejsou tak vidět okraje a obraz působí, jako by byl stejně tak velký jako komponenta samotná. Vyšší vrstva obsahuje obdélníky pro nastavení oblastí Window a ROI. Bylo však nutné vyřešit situaci, kdy jsou obdélníky blízko sebe a bylo by nesnadné trefit se na čáru konkrétní oblasti. Obdélník je proto nutné předem aktivovat kliknutím na jeho název v pravé části okna. Aktivní obdélník má zelenou barvu, neaktivní šedou a pouze s aktivním obdélníkem je možné manipulovat. V nejvyšší vrstvě komponenty je pak čára řezu. S tou je možné pohybovat za jakékoliv situace.

Kromě výše zmíněných údajů je v rámci záložky Image možné nastavit i parametry doba integrace (Time of Integration) a výkon laseru (Laser Power). K těmto účelům slouží posuvníky v pravé horní části okna. Jejich změna se vždy projeví až po uvolnění tlačítka myši. Obraz je možné také uložit do souboru stisknutím tlačítka Save Image. Na disku je pak vytvořen BMP soubor, jehož název obsahuje časový otisk okamžiku, kdy byl vytvořen. Výsledná podoba záložky Image je zobrazena na obrázku 4.5.

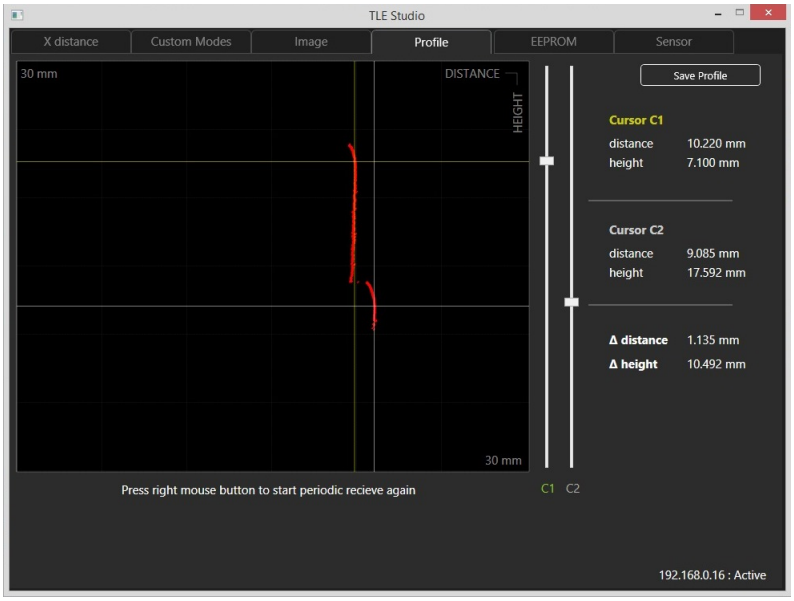


Obrázek 4.5: Záložka Image aplikace TLE Studio

4.1.6 Záložka Profile

Tato záložka slouží k zobrazení profilových dat, tedy předzpracovaných obrazových dat. Hlavním prvkem záložky je graf zobrazující profilová data. Graf reprezentuje soubor dat, který je čten ze senzoru ve formě bajtového pole. Toto pole obsahuje vždy údaj o vzdálenosti a hned za ním údaj o výšce. Je tak vytvářen dvourozměrný graf se vzdáleností na ose X a výškou na ose Y. I u tohoto grafu bylo potřeba vyřešit problém s pozicováním výsledku. Tentokrát ale výsledná pozice není závislá na celé oblasti Window, ale pouze na jeho výšce.

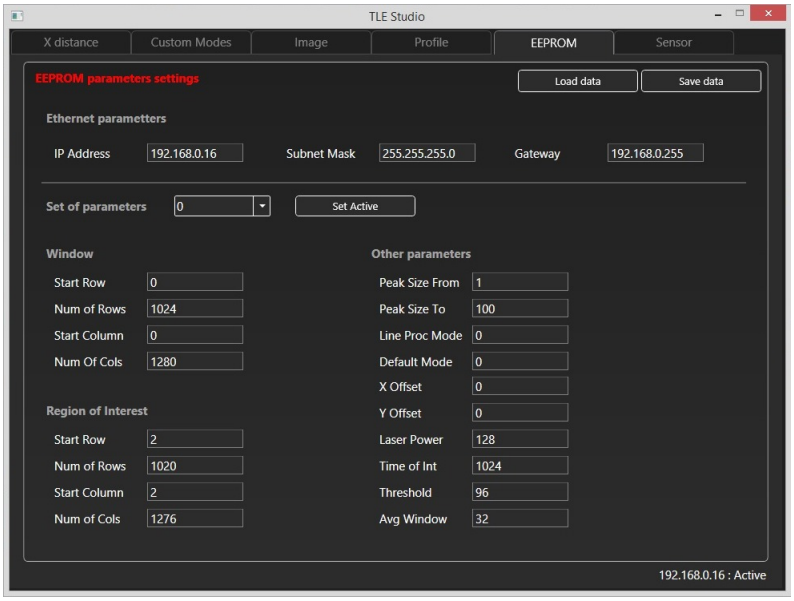
Protože se jedná o dvourozměrné pole reprezentující sadu měřených dat, byla uživateli poskytnuta možnost změřit hodnoty vzdálenosti a výšky v jednotlivých bodech použitím posuvníků na pravé straně. Při přesunutí posuvníku do oblasti měřených dat jsou vykresleny dvě čáry, které se protnou právě v bodě, který chce uživatel změřit. Použitím druhého posuvníku lze změřit i rozdíl hodnot mezi dvěma body. Pro lepší přehlednost je jeden posuvník bílý a druhý žlutý. Výslednou podobu záložky Profile zachycuje obrázek 4.6.



Obrázek 4.6: Záložka Profile aplikace TLE Studio

4.1.7 Záložka EEPROM

Hlavním účelem záložky EEPROM je umožnit uživateli načítat a ukládat data z a do vnitřní paměti senzoru. Po aktivaci záložky jsou do všech polích načteny hodnoty aktuálně zvolené stránky paměti EEPROM. Údaje lze měnit, uložit a následně i manuálně znovu načíst. Všechna pole jsou typu TextBox_Improved. Data jsou tedy před uložením validována. V rámci této záložky lze současně i nastavit aktuálně používanou stránku paměti pomocí tlačítka Set Active. Podoba záložky EEPROM je na obrázku 4.7.



Obrázek 4.7: Záložka EEPROM aplikace TLE Studio

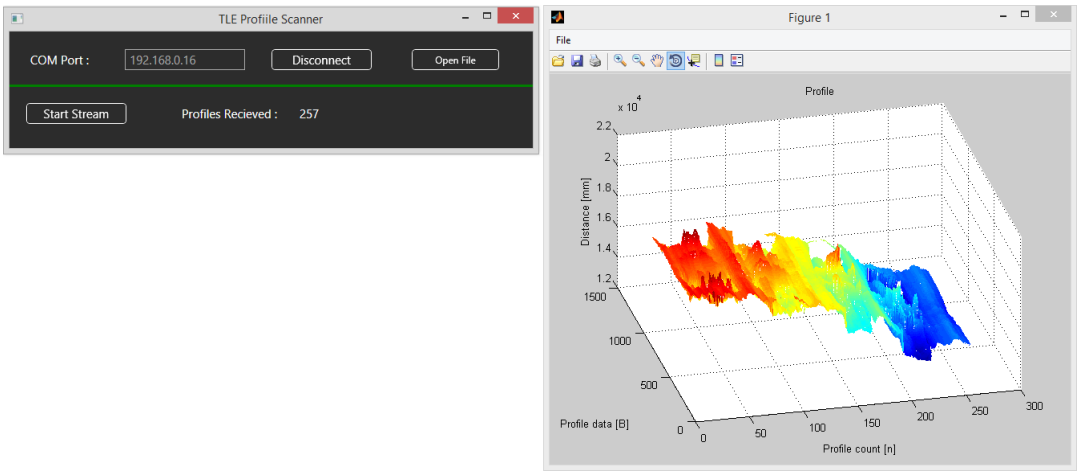
4.2 Profile Scanner

Aplikace Profile Scanner byla vytvořena za účelem prezentace možností senzoru v rámci Technické univerzity v Liberci. Tato aplikace by mohla být začleněna do TLE Studia, ale firma Metralight tuto funkcionalitu nepotřebuje. Bylo proto rozhodnuto o oddělení této části do samostatné aplikace.

Aplikace je nazvána Profile Scanner, protože využívá profilová data senzoru, ze kterých vytváří prostorový obraz objektu. S profily na ose X, bajty profilu na ose Y a vzdáleností na ose Z.

Senzor právě pro využití v podobných případech umožňuje kontinuální streamování dat (viz. kapitola 2.3.3). Zasláním příkazu pro start streamu začne senzor posílat profily tak rychle, jak sám zvládne. Jediné omezení rychlosti by tak mohlo nastat na straně počítače, který profilová data zpracovává. Aby k tomu však nedocházelo, jsou profilová data vyčítána ze senzoru a ukládána do zásobníku. Data jsou zpracována až v okamžiku, kdy uživatel stiskne tlačítko pro konec streamování dat.

Výslednou podobu programu zobrazuje obrázek 4.8. Aplikace samotná je tvořena pouze jedním oknem, ve kterém jsou prvky pro připojení k senzoru a spuštění kontinuálního streamu dat. Postup streamu je indikován popiskem, který zobrazuje počet aktuálně načtených streamů. Chce-li uživatel vizualizovat předem uložený soubor profilových dat, může využít tlačítko Open File.

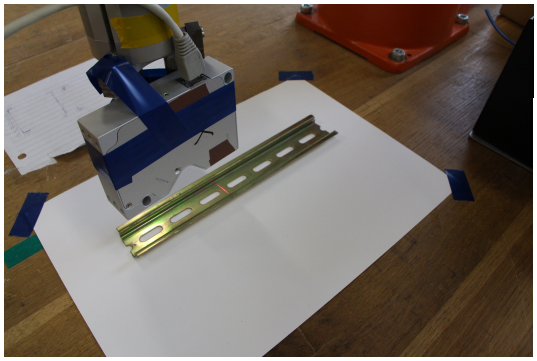


Obrázek 4.8: Aplikace Profile Scanner

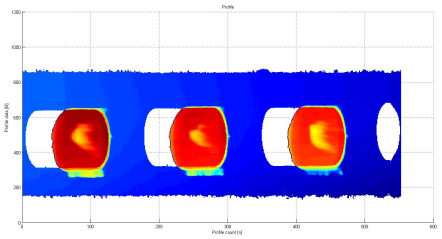
První návrhy vizualizace profilových dat počítaly s vytvořením funkce, která tato data vykreslí. Protože by ale bylo velmi složité vytvořit funkci, která by data zobrazila prostorově, byla vytvořena funkce, která profilová data zobrazovala pouze

dvourozměrně. Rozdílné vzdálenosti pak byly odlišeny různou barvou. Toto řešení se však ukázalo jako málo názroné a pro další využití nešikovné. Po prostudování několika grafických frameworků jako je například AForge.NET bylo rozhodnuto, že se pro vykreslení grafu využije prostředků programovacího prostředí Matlab, který má vykreslovací a modelovací funkce na velmi pokročílé úrovni. S objektem je možné otáčet, zvětšovat a zmenšovat velikost a výsledek je možné uložit do několika grafických souborů. Matlab podporuje načítání DLL knihoven vytvořených programovacím jazykem C# .NET. Není tak problém v Matlabu knihovnu použít. Program Profile Scaner tedy pracuje tak, že data načítá a ukládá program napsaný v jazyce C# a až při vizualizaci je zavolána funkce prostředí Matlab.

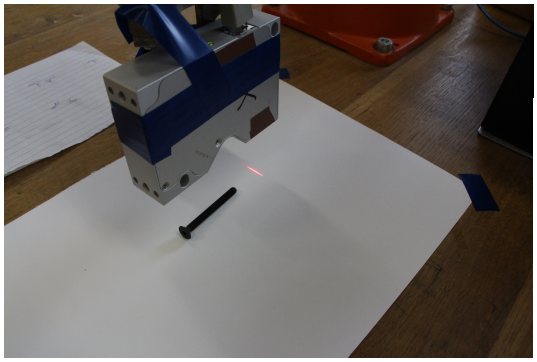
V rámci hledání možnosti využití senzoru pro účely Technické univerzity v Liberci bylo nasnímáno několik různých objektů. Senzor je například možné použít pro posuzování kvality plošných spojů a osazení součástkami. Zde se ale projevuje limitace pricipu senzoru. Senzor měří vzdálenost vysíláním laserového paprsku. Pokud je tedy deska plošných spojů lakována nebo se měď leskne, senzor není schopen přesně měřit, protože měření ovlivňují odrazy. Další možné využití je například při posuzování kvality závitů šroubů. Tam senzor vrací dobré výsledky, protože se nejedná o souvislou lesklou kovovou plochu. Na výsledném grafu jsou tedy vidět závity a v případné vady by se v grafu jasně projevily. Kromě těchto dvou případů lze senzor využít pro měření otvorů v různých materiálech či reliéfů. Výsledná měření jsou na obrázku 4.9



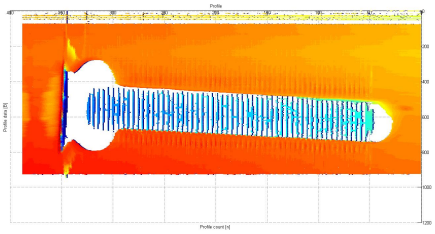
(a) Měření kovového profilu



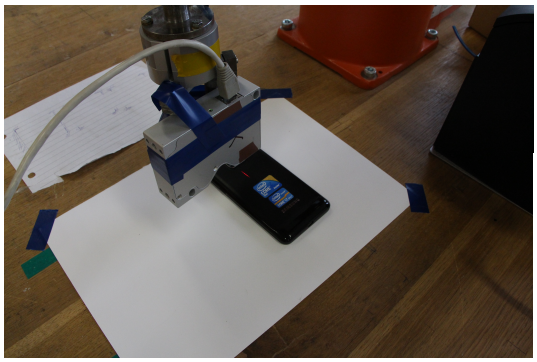
(b) Výsledný profil



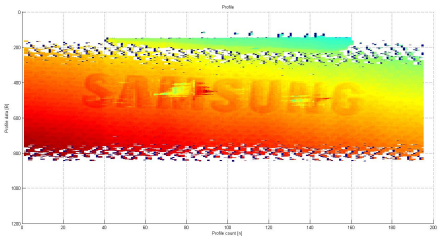
(c) Měření šroubu



(d) Výsledný profil



(e) Měření zadní strany mobilního telefonu



(f) Výsledný profil

Obrázek 4.9: Nasnímané objekty programem Profile Scanner

5 Závěr

V diplomové práci se podařilo splnit všechny body zadání. Jako první byl prostudován princip práce senzoru TLE1 a jeho komunikační protokol. Následně bylo realizováno softwarové rozhraní. Toto rozhraní bylo navrženo natolik variabilně, že je v budoucnu snadno rozšiřitelné o nové komunikační rozhraní, funkce a parametry senzoru. Rozhraní zároveň poskytuje veškerou funkcionalitu senzoru a pro programátory tak představuje snadný prostředek pro tvorbu vlastních aplikací. Pro prověření schopností rozhraní byly vytvořeny dvě aplikace. První aplikace poskytuje obecný přehled o funkcích senzoru díky vizualizaci jeho dat. Uživatel tak má snadný přístup k měřeným hodnotám a obrazovým a profilovým datům. Zároveň může snadno pracovat s vnitřní pamětí a nastavovat parametry senzoru.

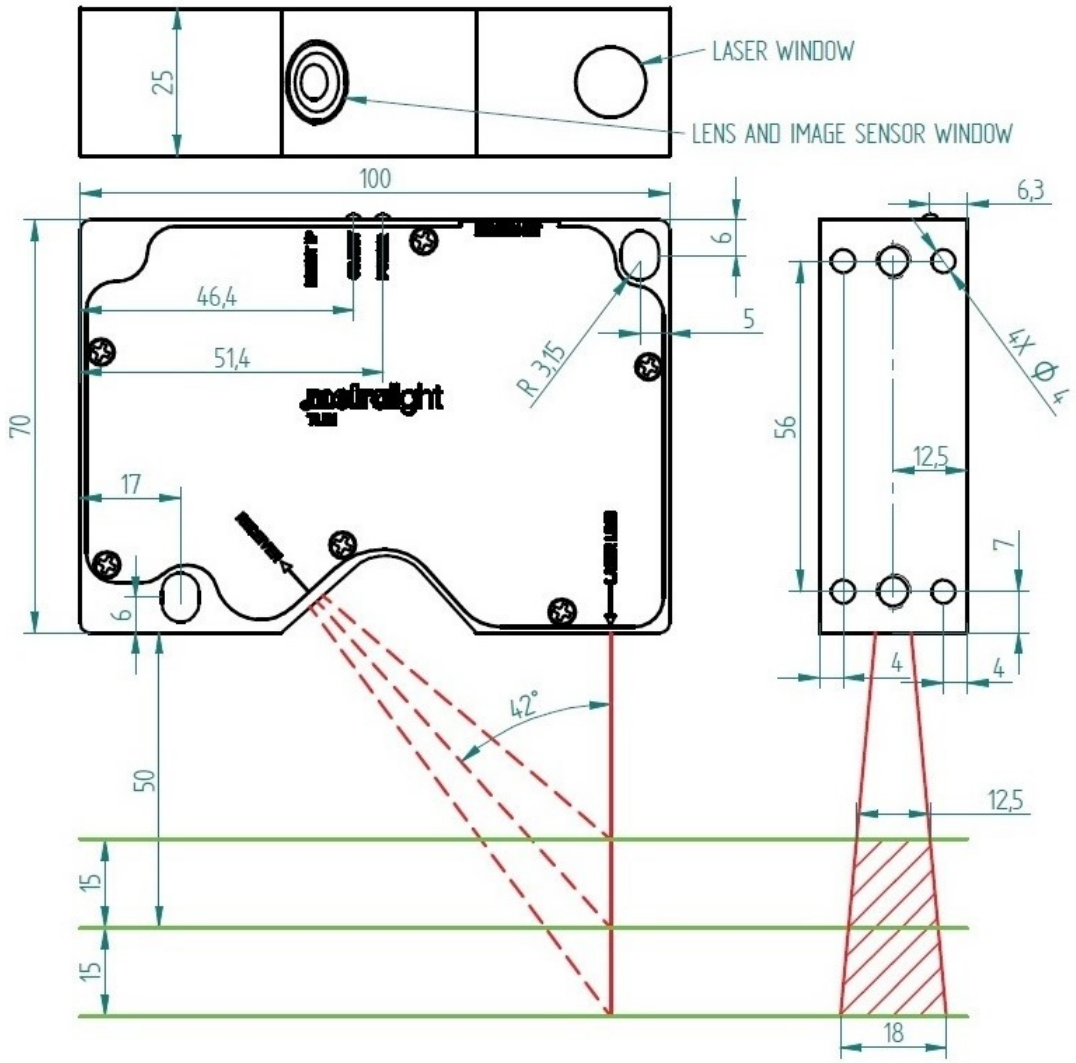
Rozhraní je možné v budoucnu dále rozšířit například přidáním připravených grafických komponent, které programátorovi umožní data vizualizovat přetažením těchto komponent do svého programu. Je také možné do rozhraní začlenit další senzory firmy Metralight, protože většina z ních sdílí stejné komunikační rozhraní a liší se pouze v parametrech. V aplikaci Profile Scanner by do budoucny bylo možné začlenit funkce pro měření údajů přímo v obraze větším využitím programovacího prostředí Matlab.

Literatura

METRALIGHT, 2012. *TLE1 User's Guide*. Liberec. Dostupné z: http://www.metralight.com/products/doc/tle1/tle1_ug_d.pdf.

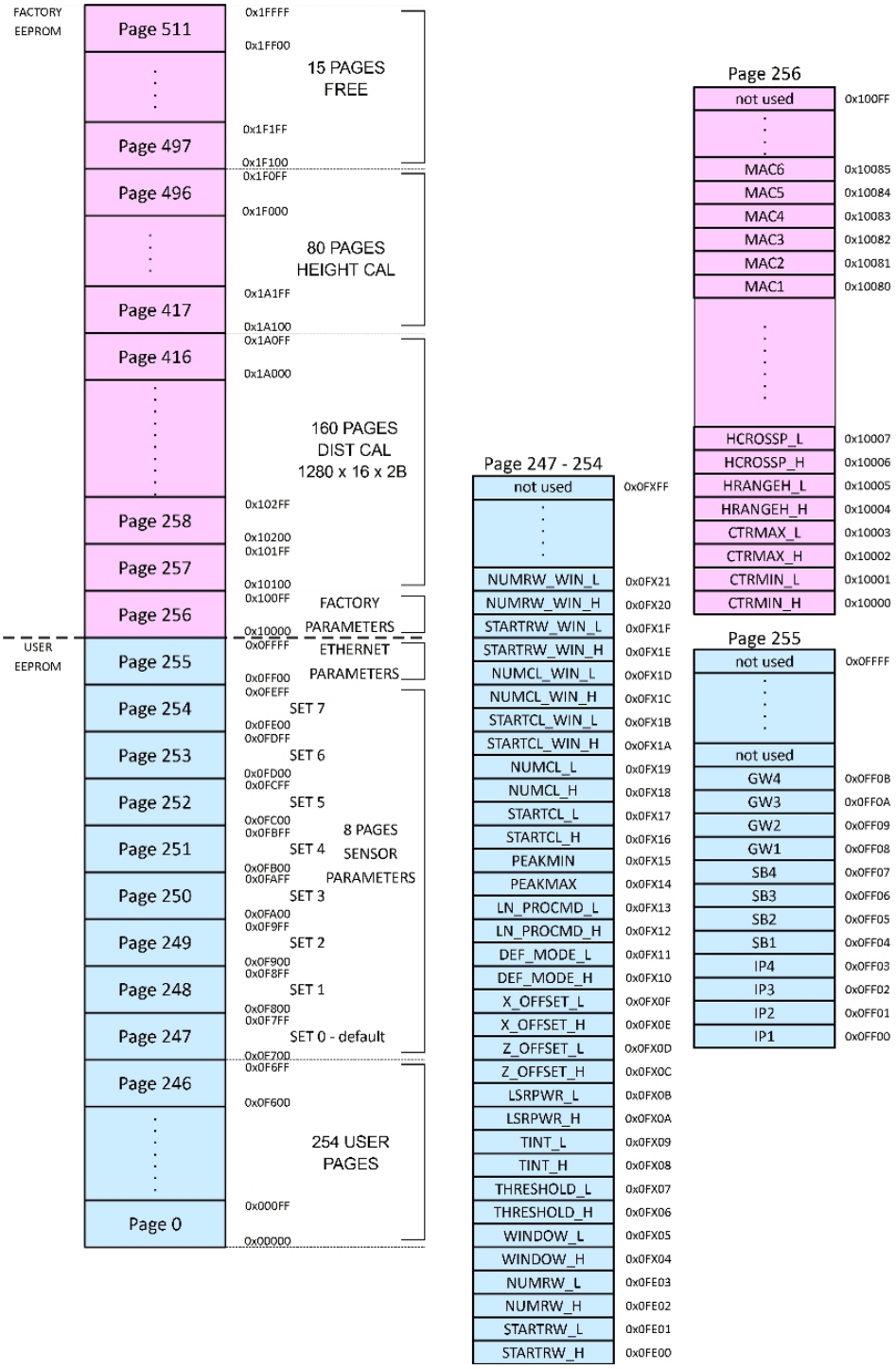
SHARP, J., 2010. *Microsoft Visual C# Krok za krokem*. Brno: Computer Press. ISBN 978-80-251-31473.

A Rozměry senzoru TLE1



metralight

B EEPROM mapa



C Volatilní parametry senzoru

Parametr	Popis	Čtení	Zápis	Rozsah
ROI_START_COLUMN	První sloupec ROI	0x50	0x58	2 – 1248
ROI_NUM_OF_COLUMNS	Počet sloupců ROI	0x51	0x59	32 – 1278
WIN_START_LINE	První řádek Window	0x52	0x5A	1 – 1020
WIN_NUM_OF_LINES	Počet řádků Window	0x53	0x5B	4 – 1024
WIN_START_COLUMN	První sloupec Window	0x54	0x5C	1 – 680
WIN_NUM_OF_COLUMNS	Počet sloupců Window	0x55	0x5D	600 – 1280
ROI_START_LINE	První řádek ROI	0x80	0x88	2 – 1022
ROI_NUM_OF_LINES	Počet řádků ROI	0x81	0x89	2 – 2022
LINE_PROCMODE	Režim procesování	0x86	0x8E	0 – 3
PEAK_SIZE	Hodnota procesování	0x87	0x8F	1 - 65535